UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable) EC	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO.; PROJECT NO.; TASK NO.; WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
IMPLEMENTATION OF FUZZY INFERENCE SYSTEMS USING NEURAL NETWORK TECHNIQUES

12. PERSONAL AUTHOR(S)
HUDGINS, Billy Eugene, Jr.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM ___ TO ___	1992 March	46

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	fuzzy inference, neural networks, adaptive training, gradient descent, membership functions

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Fuzzy inference systems work well in many control applications. One drawback, however, is determining membership functions and inference control rules required to implement the system, which are usually supplied by ‘experts’. One alternative is to use a neural network-type architecture to implement the fuzzy inference system, and neural network-type training techniques to ‘learn’ the control parameters needed by the fuzzy inference system. By using a generalized version of a neural network, the rules of the fuzzy inference system can be learned without the assistance of experts.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL YANG, Chyan	22b. TELEPHONE (Include Area Code) (408) 646-2266; 22c. OFFICE SYMBOL EC/Ya

DD FORM 1473, 84 MAR   83 APR edition may be used until exhausted. All other editions are obsolete   SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED

Implementation of Fuzzy Inference Systems Using Neural Network Techniques

by

Billy E. Hudgins, Jr.
Lt, USN
B.S, Georgia Institute of Technology, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

March 1992

ABSTRACT

Fuzzy inference systems work well in many control applications. One drawback, however, is determining membership functions and inference control rules required to implement the system, which are usually supplied by 'experts'. One alternative is to use a neural network–type architecture to implement the fuzzy inference system, and neural network–type training techniques to 'learn' the control parameters needed by the fuzzy inference system. By using a generalized version of a neural network, the rules of the fuzzy inference system can be learned without the assistance of experts.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to express my gratitude and appreciation to the faculty and staff of the Electrical and Computer Engineering department for providing me with the opportunity and encouragement to explore many exciting facets of electrical engineering. I would like to offer special thanks to Chyan Yang for his assistance and Jon Butler for his advice. Also, I would like to thank Jyh-Shing Jang for supplying data used in the simulations. Finally, I would like to thank my wife Cindy for her patience during the compilation of this thesis.

I. INTRODUCTION

The recent emergence of artificial neural networks and fuzzy inference systems have encouraged many designers to combine the two concepts into a single system having characteristics superior to either system alone [Ref. 1, 2, 3, 4, 5, 6]. One method of tying the two systems together is by using the learning capabilities of an artificial neural network–type architecture (ANN) to describe and tune the membership functions and inference rules of a fuzzy inference system. Thus, by using a connectionist approach, where the nodes function to implement parts of the fuzzy inference paradigm, the expert needed to describe the control parameters can be eliminated. Thus, by using neural network training techniques, the designer is relieved of the need for experts to delineate explicit rules governing the fuzzy inference system, since the system itself is capable of defining these rules.

The association of neural network and fuzzy inference systems gives rise to designs with very different topologies and functionalities. Therefore, to better understand how these two concepts can be merged, an implementation of a system combining the two is described in Chapter II. The results of simulations using this system are given in Chapter III.

An overview of some of the concepts of neural networks and fuzzy inference systems can be found in Appendix A. The MATLAB code to implement the system discussed in Chapter II is given in Appendix B. Graphical results of the simulations discussed in Chapter III are given in Appendix C.

II. CONNECTIONIST FUZZY INFERENCE SYSTEM

The application of fuzzy inference systems for the control of systems, as well as neural networks for the classification and identification of patterns, is well documented [Ref. 7, 8, 9, 10]. This chapter introduces the concept of combining these two paradigms to provide a connectionist approach to implementing a fuzzy inference system.

A. GENERALIZED NEURAL NETWORK (GNN)

One way to utilize the learning capabilities of a neural network is to teach a control system the proper associations between inputs and outputs. However, the level of interaction between the processing elements in the various layers is at such a fundamental level that no exploitable information can be gathered from the parameters within the network, thus providing only a black box processing approach for implementing the input/output relationships. On the other hand, a fuzzy inference system allows a few simple rules to control a process with very good results. However, the fuzzy system must be supplied with these rules, which are usually obtained from 'experts', either by careful observation of their actions, or by linguistic interpretation of their actions.

For very complicated processes, rule extraction is inappropriate, either because there is no expert to mimic, or because the actions of the expert are so complex that they defy linguistic explanation. In this case, an alternative must be found.

One method of achieving the self-adjustment of rules in a fuzzy inference system is suggested by Jang [Ref. 1]. A generalized network is set up which includes fuzzy inference features. After it is determined how many membership functions the inputs will be assigned, thereby determining the dimensionality of the network, a training data set is used to adjust the parameters that determine the shape of these membership functions, as well as the manner in which the rules will be combined to give the overall results. Jang only considers MISO (multiple input, single output) cases, but this is sufficient to describe many useful control situations.

Each layer in the generalized network provides nodes which take the outputs from previous layers and provides inputs for succeeding layers. This architecture has many similarities with feedforward artificial neural network (ANN) implementations, and the functions exhibited by the nodes in each layer could be implemented with subnets of ANN's [Ref. 11]. Nonetheless, the generalized implementation allows direct access to the rules produced by the training of the network, which can be exported to other, more standardized, implementations of fuzzy inference control systems. Figure 2.1 gives an example representation of a generalized neural network (GNN) with two inputs, one output, and two membership functions per input. Each membership function represents a linguistic attribute for each input, in this case 'high' and 'low.'

The following is a functional description of each of the layers in a GNN:

Layer 1 For every distinct input used in the system, a number of membership function nodes are added. The output of these nodes represent the degree to which the input belongs to the linguistic, or fuzzy, variable represented by that node. For example, the Layer 1 nodes connected to each input might correspond to the linguistic terms 'small' and 'large'. Thus, the outputs from each of these nodes would indicate how well the input 'fit' within each of these categories. The shapes of the membership functions are variable and parameters used to adjust them are known as premise parameters, and are determined during the training process. The function used to express the membership functions in this layer is

$$O^1 = \frac{1}{1 + [(\frac{x-c}{a})^2]^b}, \tag{2.1}$$

where x is the input value and $\{a, b, c\}$ are parameters for the membership function for a particular Layer 1 node.

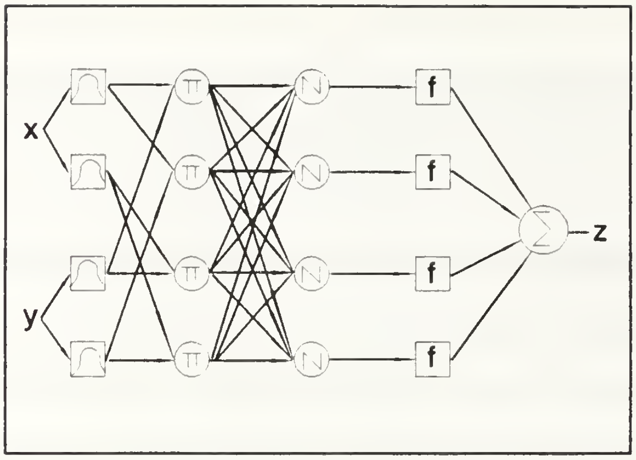

Figure 2.1: A GNN Fuzzy Inference System

Layer 2 The outputs of each of the Layer 1 nodes are combined with outputs from the other Layer 1 nodes so that every combination of outputs from different membership functions within groupings for a particular input are available at the next layer. The output from the Layer 2 nodes represent the absolute firing strength of a particular fuzzy rule made by combining the linguistic variables from different inputs.

Layer 3 Each node in this layer represents a particular firing rule. The absolute firing strength received from Layer 2 is normalized against the sum of absolute firing strengths from all rules, and this value is sent to Layer 4.

Layer 4 The nodes in this layer weight the relative firing strength from the previous layer by a biased combination of the input values, and passes this value on to Layer 5. The functional description for the output of a node in this layer for the example in Figure 2.1 would be

$$O^4 = w_n * (d_x * x + d_y * y + d_0), \tag{2.2}$$

where w_n is the output from Layer 3, x and y are the inputs to the network, and $\{d_x, d_y, d_0\}$ are parameters that will be adjusted during the training process, called consequence parameters.

Layer 5 All Layer 4 outputs are summed and sent through as the output of the network.

The purpose of the entire network is to create a system whereby certain parameters can be adjusted so that an appropriate output will be realized for a given input. These parameters are found in Layer 1 and Layer 4, and are adjusted using training data that give the correct output for a given input vector. The manner in which these parameters are actually adjusted, or 'trained,' involves the use of a gradient descent algorithm that minimizes a squared energy error function between the correct 'target' output and the actual output of the network for several training samples. The gradient descent method for training the parameters gives the connectionist fuzzy inference system attributes similar to ANN's, which employ similar gradient descent techniques utilizing a variant of this method called backpropagation for training.

B. IMPLEMENTATION OF THE GNN

The implementation of the GNN was coded using MATLAB, a high level language ideally suited for scientific applications. The actual code is given in Appendix B. The results of several simulations are given in Appendix C and explained in Chapter III.

The algorithm can be divided into two parts. The first is the calculation of the outputs of the nodes holding the Layer 1 parameters constant and adjusting the Layer 4 parameters. All of the input vectors are presented to the input layer, and the output of all nodes up to Layer 4 are calculated. Using the built-in MATLAB function *pinv*, which calculates the pseudoinverse of a matrix, the inputs to Layer 4 and the desired outputs of Layer 4, backtracked from the desired outputs, are used to determine the Layer 4 parameters. Using these parameters, the outputs of the nodes of the last two layers are determined. Thus, the Layer 4 parameters are optimized for the entire training data set with the Layer 1 parameters held constant.

Furthermore, with the Layer 4 parameters held constant, the parameters in Layer 1 are adjusted to minimize the error measure defined by Equation 2.3 for each input vector, where E_p is the error measure for the p-th input vector, T_P is the corresponding desired output, and O_p^5 is the actual output.

$$E_p = (T_p - O_p^5)^2. \tag{2.3}$$

Thus, for a training data set containing P samples, there will be P component error measures, and therefore the total error measure will be given by $E = \sum_{p=1}^{P} E_p$. In order to minimize this total error measure, E, a gradient descent method was used to determine the adjustment of the parameters in Layer 1.

In order to implement this gradient descent method, the manner in which E changes with respect to each of the parameters in Layer 1 had to be determined. Because the functional descriptions between the different layers and nodes were readily available, the straightforward approach of taking partial derivatives of E with respect to α, a particular Layer 1 parameter, was utilized.

The first step in determining this partial derivative was by invoking the chain rule on Equation 2.3. The first two steps of this procedure are given in Equations 2.4 and 2.5.

$$\frac{\partial E_p}{\partial \alpha} = \frac{\partial E_p}{\partial O_p^5} \cdot \frac{\partial O_p^5}{\partial \alpha}. \tag{2.4}$$

$$\frac{\partial E_p}{\partial O_p^5} = -2(T_p - O_p^5). \tag{2.5}$$

Thus continuing, $\frac{\partial O_p^5}{\partial \alpha}$ was determined by considering the structure of the GNN, and taking partial derivatives to determine the relationships in the chain rule back to Layer 1. Consequently, this backward determination of the error at each node back to the input layer is similar to the backpropagation technique used for training ANN's, although the partial derivatives must be determined at each layer since the nodes in different layers may have different transfer functions.

Nevertheless, eventually the chain rule gets to the point where $\frac{\partial O_p^1}{\partial \alpha}$ must be determined. Although each node in Layer 1 will give a different result when α is a, b, or c, the general relationships can be defined by Equations 2.6, 2.7, and 2.8.

$$\frac{\partial O_p^1}{\partial a} = \frac{2}{ab} \cdot \left(\frac{1}{1 + [(\frac{x-c}{a})^2]^b}\right)^2 \cdot \left(\frac{x-c}{a}\right)^2. \tag{2.6}$$

$$\frac{\partial O_p^1}{\partial b} = -2\log\left(\frac{x-c}{a}\right)\cdot\left(\frac{1}{1+[(\frac{x-c}{a})^2]^b}\right)^2\cdot\left[\left(\frac{x-c}{a}\right)^2\right]^b. \quad (2.7)$$

$$\frac{\partial O_p^1}{\partial c} = \frac{2b}{a}\cdot\left(\frac{1}{1+[(\frac{x-c}{a})^2]^b}\right)^2\cdot\left(\frac{x-c}{a}\right)^2. \quad (2.8)$$

In these equations, a,b,c are the Layer 1 parameters for a particular node, and x is the input to that node.

Thus, with $\frac{\partial E_p}{\partial \alpha}$ known, the total error measure partial derivative can be determined,

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^{P}\frac{\partial E_p}{\partial \alpha}. \quad (2.9)$$

Furthermore, the gradient descent algorithm updates α, the Layer 1 parameter, after each training cycle, giving

$$\Delta\alpha = -\eta\frac{\partial E}{\partial \alpha}, \quad (2.10)$$

where η is a proportionality constant given by

$$\eta = \frac{k}{\sqrt{\sum_\alpha (\frac{\partial E}{\partial \alpha})^2}}. \quad (2.11)$$

In Equation 2.11, k is the step size of the gradient descent algorithm. In the GNN algorithm, k is adjusted by observing how the error measure E changes after each iteration of training. Initially, k is set to unity, and increased by 10% if the error measure decreases for four consecutive training cycles, and is decreased by 10% if the error measure alternates increasing and decreasing twice in a row. These two cases indicate that E is converging or oscillating, and thus k should be increased to speed convergence, or decreased to prevent oscillation, respectively.

III. SIMULATION RESULTS OF THE GNN

Three sample problems were used to verify the performance of the GNN. The first two problems had two inputs and a single output, while the third had three inputs and a single output. The sample data used in the first example is listed in Table C.1, and it represented actual data taken from a fuzzy inference control system designed in a conventional manner [Ref. 1]. Examples 2 and 3 were efforts to infer the highly nonlinear functions given by Equations C.1 and C.2. The target output surfaces for the first two examples are shown in Figures C.1 and C.2. Since the third example described a four-dimensional hyperspace, it was not shown.

To provide flexibility in utilization, certain parameters in the algorithm for the GNN can be altered. The most important parameter in the network configuration is the number of membership functions in Layer 1. Although increasing the number of membership functions increases the complexity of the overall network, the overall performance is also increased. To show the difference that altering this parameter makes, the three examples were trained using different numbers of nodes, and thus membership functions, in the first layer. The results of these simulations are shown in Figures C.3 through C.36.

The input for the first data set ranged from -10 to 10 and were selected at two unit intervals. Thus, considering both inputs, a total of 121 samples were used. The data for the second example used a similar partitioning scheme for the input data, and also used 121 data points. The final example was sampled uniformly over all three inputs in the range [1,6] in unit increments, for a total of 216 samples.

As a measure of how well the network learned the input/output relationships, an error measure comparing the actual output and desired output, called the average percentage error (APE), was used. Two slightly different computations of the APE are given by Equations C.3 and C.4. The second version, APE_3, was used on the third example, since it had no zero outputs. The first version, APE_{12}, was applied to the first two examples, since their output range included zero, and thus the second, more accurate, version would not be suitable.

Comparing the results of simulations for the first data set using two, three, and four membership functions (MF's) per input, the APE graphs show that for two MF's, the APE stays greater than 28% past 200 iterations of the training data (Figure C.6). For three membership functions, the APE figure is greatly reduced to around 2% for 200 iterations (Figure C.10). However, using four MF's per input, the APE index dropped down to slightly above 0.2% (Figure C.14).

Similar results can be found for the second and third data sets, and Table 3.1 illustrates how the APE value is decreased for increased numbers of membership functions. The table reflects the APE value after 200 iterations for the first two sample data sets, and after 50 iterations for the third data set.

TABLE 3.1: APE VALUES FOR EXAMPLES 1,2, AND 3

	2 MF's	3 MF's	4 MF's
Ex.1	28.45	1.69	0.19
Ex.2	15.12	0.55	0.10
Ex.3	0.04	0.00	——

However, the enhanced performance of the network gained by increasing the number of membership functions per input does have a cost. The computational cost, whether measured in time or floating point operations, is the major limitation of the GNN algorithm. Table 3.2 shows the number of millions of floating point operations (MFlop's) that MATLAB reported during the running of the algorithm

TABLE 3.2: MFLOP'S REQUIRED FOR SIMULATION RESULTS

	2 MF's	3 MF's	4 MF's
Ex.1	216.2	612.4	1,439.1
Ex.2	216.8	607.7	1,420.5
Ex.3	450.4	1,822.5	———

that gave the results in the previous table. It is readily apparent that the number of membership functions should be minimized to reduce the computational cost, but the accuracy of the network must be considered in determining this lower bound.

IV. CONCLUSIONS

Neural networks provide a very good method of identifying and classifying patterns of data. Fuzzy inference provides a natural background by which control rules can be expressed in a general sense. The use of neural network techniques in the generation of the fuzzy inference control rules provides a means to implement the fuzzy paradigm without interjection of an expert to describe its rules. However, by generalizing the network to produce these self–generating rules, the complexity of the training algorithm is increased, thereby increasing overall complexity, and incurring increased computational costs. However, since the learning phase of the generalized neural network should only be required once, or at least after long time intervals, the increased training time might be a small price for the increased flexibility and simplicity found in the fuzzy inference control architecture.

APPENDIX A: INTRODUCTION TO NEURAL NETWORKS AND FUZZY INFERENCE

This appendix presents some of the fundamentals behind neural networks and fuzzy inference systems that will enable better understanding of the preceding chapters.

A. NEURAL NETWORKS

In an attempt to exploit the millions of years of evolution that nature has had to develop the human brain and nervous system, scientists and engineers have attempted to use some of the basic fundamentals of the networking of neural components within the human body in order to produce artificial neural networks (ANN's). Although an ANN is a very simplistic abstraction of a real neural network, say of the brain, it has been used successfully in many areas [Ref. 12]. To say the least, the idea of using a neural net paradigm that has had millenia to evolve has created much interest in most branches of science.

The fundamental concept in an ANN is the idea of neurons connected together to form a network, with inputs processed at one end, and output generated at the other. Figure A.1 shows a generic artificial neuron. The makeup of this neuron consists of a set of inputs from a previous layer of the network, a weighting operation on each of these inputs, a summation of these weighted inputs, and a filtering operation through a sigmoid function. Finally, this output is sent on to the next layer.

The sigmoid function acts as a normalization, or activation function, for the summation of weighted inputs. Thus, the output of one node always falls within a predetermined range. This is important since this allows a stabilizing effect to be

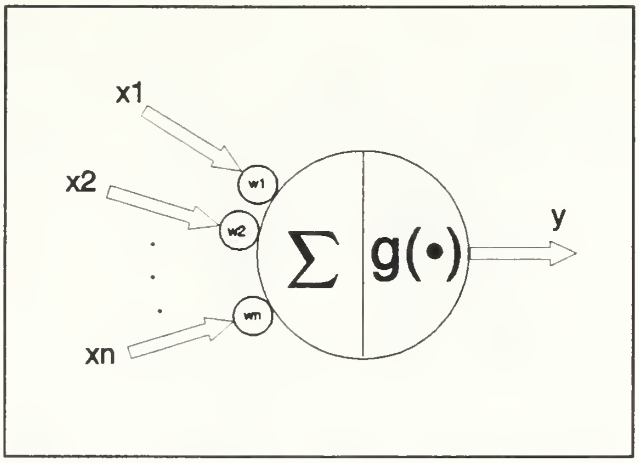

Figure A.1: Generic Artificial Neuron

asserted on the network at all times.

Probably the most important aspect of an artificial neuron is the weighting of the inputs to the neuron. Since these weights are variable, different assignments of values to these variables cause the neuron to act in very different ways. In fact this is the driving force behind the use of ANN— with a sufficient number of elements within a neural network, any nonlinear input/output function can be 'learned' by adjusting these weights.

The topology of connecting neurons, the shape of the activation function, and the manner of adjusting the weights are far too numerous to describe within this paper. For a more detailed introduction to neural networks see [Ref. 9].

However, the types of neural networks can be classified as *feedforward* networks, which only have one-way connections between layers, *feedback* networks, which can be fully connected. An example of a feedforward network with an input layer with four inputs, one hidden layers with three nodes, and an output layer with four nodes is shown in Figure A.2. The reason for the term 'hidden' is because a neural network is usually considered a black box, and thus access is only available to the input and

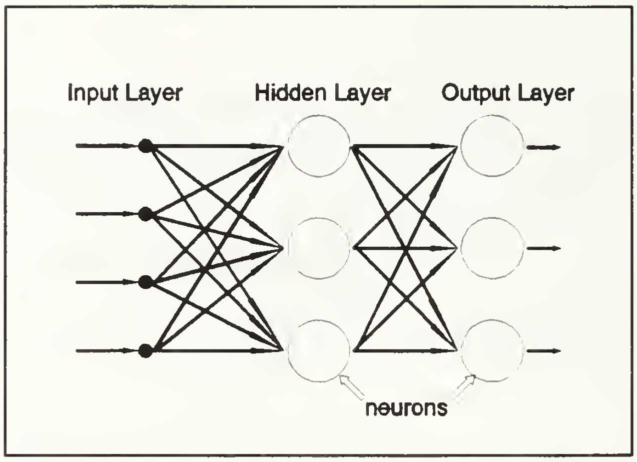

Figure A.2: A Feedforward ANN

output layers. Therefore, all other layers in between are 'hidden' from the user.

One of the most widely used methods of updating weights on the input links to the processing elements in the network is the backpropagation method. This method uses an iterative gradient descent algorithm designed to minimize the mean square error between the actual and desired outputs of the network [Ref. 9].

B. FUZZY INFERENCE

Fuzzy logic is an attempt to organize data and relationships between data in a way that is intuitive and natural. The essential building block in fuzzy logic is the concept of membership functions. Simply put, a membership function is a nonlinear mapping of a variable into a likelihood that the value of the variable will be found in the set of values that the membership function represents. An example of a mapping of this type can be seen in Figure A.3 [Ref. 13]. Here systolic blood pressure is the variable to be mapped, and five different mappings of this variable, representing five different membership functions, are shown. By convention, the output from a membership function is usually expressed in the range of the closed interval [0,1].

Furthermore, each membership function is assigned a linguistic term that de-

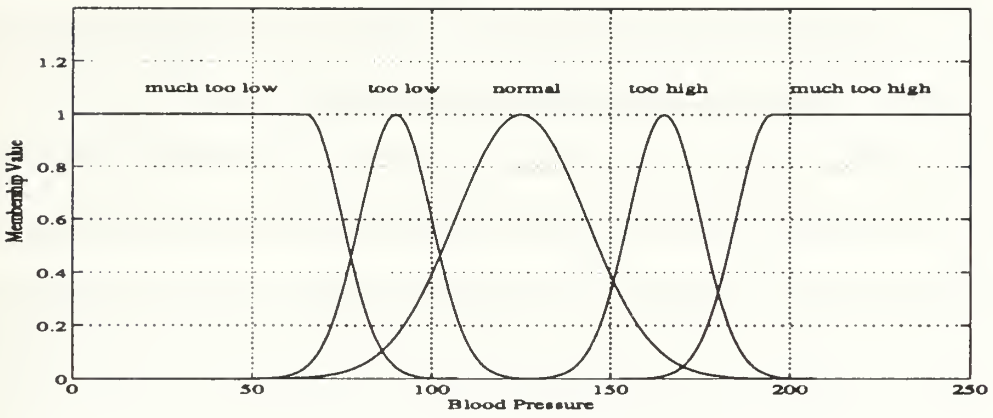

Figure A.3: Membership Functions for Systolic Blood Pressure

scribes the relationship between the set of values represented by that membership function and those of all other membership functions under consideration. In this example, the linguistic terms applied to the input variable of systolic blood pressure are 'normal', 'too high', 'too low', 'much too high', and 'much too low.' The output of the membership functions for a certain value of the input variable relates how well that value of the variable 'fits' the linguistic term that represents each membership function. A membership likelihood value close to zero indicates that the input value has a low likelihood of belonging to the set of values that best describes the linguistic term representing the membership function. A likelihood value close to one would indicate a high degree of association of the input value with the set of values that best describe that linguistic variable.

As an example, a systolic blood pressure of 125 would have a likelihood very close to one for the linguistic term 'normal', indicating a high correlation between a pressure of 125 and a 'normal' pressure. On the other hand, a pressure of 100 would have a likelihood of 0.5 for both the linguistic terms 'too low' and 'normal'. This indicates that, for this example, a systolic blood pressure of 100 could be diagnosed as either 'too low' or 'normal' with equal likelihood. Obviously, there is some arbitrariness as to the shape and placement of these membership functions, and indeed this has been a disadvantage in implementing fuzzy logic controllers [Ref. 7, 8].

A more abstract interpretation of membership functions is that an arbitrary variable— such as current from an alternator, temperature of a heat sink, or weight of the contents of a variable ballast tank— can be represented by several overlapping linguistic, or fuzzy, terms. For each value of the input variable, which is continuous, a value in the range [0,1] is assigned to each of the linguistic terms as a measure of the relationship that the input has to that linguistic term. So, instead of comparing the value of the input variable, for example, as either 'warm' or 'cool', the input is said to have the attribute 'warm' to a certain degree, and to have the attribute 'cool' to a certain degree. So, for a temperature of 25°C, the membership value in the sets for 'warm' and 'cool' might be 0.5 each, but for the linguistic term 'room temperature', the output would be most likely one. All linguistic terms, and thus the membership functions they represent, provide a response for all values of the input variable, even though that response may be zero.

The second fundamental concept in implementing a fuzzy inference system is the idea of if–then statements, or inference rules. This is the method by which the system is able to effect the control laws that allow plant parameters to be maintained. The concept of if–then statements as decision makers is common, especially in expert systems and computer algorithms. However, for a fuzzy inference system there are several rules that must be considered at one time, and the output is determined by considering the overall effect of all rules taken at once.

The most common method of implementing the if–then rules is by considering methods used in Boolean logic and find similar functions that are appropriate in fuzzy logic. For the statement , " If U is A and V is B then W is D," the operation of the premise 'if', the conjunction 'and', and the implication 'then', must be implemented with fuzzy logic functions so that the intent of the overall implication will still be intact. There are several methods of constructing these functions [Ref. 14], but the most widely used treat the 'and' operation as a min(x,y), the 'if' premise as a membership operation, and the 'then' implication as a center of moments operation.

A more detailed introduction to fuzzy logic and fuzzy inference control systems can be found in [Ref. 8].

APPENDIX B: MATLAB CODE FOR THE GENERALIZED NEURAL NETWORK

The MATLAB code for the implementation of the GNN of Chapter II follows. Also included are the subfunctions that the main program calls.

```
clear;clg;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Generalized Neural Network
%
%    Implementation of an algorithm suggested by Jang.  Three data sets
%  are available, j1.dat, j2.dat, and j3.dat, corresponding to the
%  examples used in Jang's paper.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Change the following parameters depending on desired topology:
%
load j3.dat;            %May be j1.dat, j2.dat, or j3.dat
T = j3;                 %Local variable
M = 2;                  %Number of membership functions per input
epochs = 200;           %Number of training iterations using all data
                        %  in a batch mode

%  Determine all other parameters to be used and define some variables
[P,N] = size(T);        %P is number of samples, and ...
N     = N - 1;          %  N is the number of inputs.
IN    = T(:,1:N);       %Training inputs to the network
OUT   = T(:,N+1);       %Actual output
L1    = N*M;            %Number of nodes in the first layer
L2    = M^N;            %Number of nodes in layers 2,3, and 4.
I     = ones(1,L1);     %Vector used in the training phase
kk    = 1;              %Step-size parameter for updating premise parameters
Keep  = 100*ones(1,5);%Vector holding current past values of kk

%  Determine initial premise parameters used in the membership functions
for i=1:N,
  for k=1:M,
    Prems(1,(i-1)*M+k) = (max(IN(:,i))-min(IN(:,i))/(M-1));
    Prems(2,(i-1)*M+k) = 2;
    Prems(3,(i-1)*M+k) = min(IN(:,i))+(k-1)*(max(IN(:,i))-min(IN(:,i)))/(M-1);
  end;
end;
```

```
%  Store intial values of premise parameters
InitPrems = Prems(:,1:M);

%  Construct input vector
for i=1:N,
  for k=1:M,
    InPrems(:,(i-1)*M+k) = IN(:,i);
  end;
end;

%  Construct matrix used to determine Layer 2 inputs
for n=1:L2,
  for m=1:N,
      check(n,m) = floor(rem((n-1)/M^(N-m),M))+1+M*(m-1);
  end;
end;

%  Start iteration for batch processing
% Part One: Determine forward values
for t = 1:epochs
  for i = 1:P                   %Cycle through all samples of the input

% Layer One output
    O1(i,:)=I./(I+(((InPrems(i,:)-Prems(3,:))./Prems(1,:)).^2).^Prems(2,:));

% Layer Two output
    O2(i,:) = ones(1,L2);
    for n = 1:L2,
      for m = 1:N,
        O2(i,n) = O2(i,n).*O1(i,check(n,m));
      end;
    end;

% Layer Three output (weight matrices for the nine rules)
    Nsum(i,1) = sum(O2(i,:)');
    O3(i,:) = 1/Nsum(i,1) * O2(i,:);

  end;

% Determine consequence parameters
  A = [];
  for k = 1:L2
    for m = 1:N+1,
      TT(:,m) = O3(:,k);
    end;
    A = [A TT.*[IN ones(T(:,1))]];
  end;
  D = pinv(A) * OUT;
  for k = 1:L2
    Consq(:,k) = D((k-1)*(N+1)+1:k*(N+1),1);
  end;
```

```
% Layer 4 and 5 outputs
  for i=1:P,
    O4(i,:) = O3(i,:).*([IN(i,1:N) 1]*Consq);
    O5(i,1) = sum(O4(i,:)');
  end;

% Part Two: Hold consequence parameters and update premise parameters
  for p=1:P,
    for i=1:L1,
      dO1(i,:) = [deriva(Prems(:,i),InPrems(p,i))...
         derivb(Prems(:,i),InPrems(p,i)) derivc(Prems(:,i),InPrems(p,i))];
    end;
    dO2 = zeros(L2, 3*L1);
    for i=1:L2,
      for c=1:L1,
        if (member(check(i,:),c)),
          for v=1:3,
            dO2(i,(c-1)*3+v) = 1;
            for n=1:N,
              if (check(i,n)==c),
                dO2(i,(c-1)*3+v)=dO2(i,(c-1)*3+v)*dO1(c,v);
              else
                dO2(i,(c-1)*3+v)=dO2(i,(c-1)*3+v)*O1(p,check(i,n));
              end;
            end;
          end;
        end;
      end;
    end;
    dNsum = sum(dO2);
    for i=1:L2,
      K(i) = [IN(p,:) 1]*Consq(:,i);
      for c=1:L1,
        for v=1:3,
          dO3(i,(c-1)*3+v)=1/Nsum(p,1)*dO2(i,(c-1)*3+v)...
              -1/(Nsum(p,1)^2)*O2(p,i)*dNsum(1,(c-1)*3+v);
          dO4(i,(c-1)*3+v)=K(i)*dO3(i,(c-1)*3+v);
        end;
      end;
    end;
    dO5 = sum(dO4);
    dEp(p,:) = -2*(OUT(p,1)-O5(p,1))*dO5;
  end;

% Determine parameters to update premise parameters
  dE = sum(dEp)
  eta = kk/sqrt(dE*dE')
  dalpha = -eta*dE

  for i=1:L1,
    for c=1:3,
      dalf(i,c) = dalpha(1,(i-1)*3+c);
    end;
  end;

  Prems = Prems + dalf';          %Update

% Update step-size parameter
  ErrMeas = (OUT-O5)'*(OUT-O5);   %Squared error
  Keep = [Keep(2:5) ErrMeas];

%Increase step-size parameter if ErrMeas decreases four times consecutively
  if (Keep(1) > Keep(2)),
    if (Keep(2) > Keep(3)),
      if (Keep(3) > Keep(4)),
        if (Keep(4) > Keep(5)),
          kk = 1.1*kk;
        end;
      end;
    end;
  end;

% Decrease parameter if ErrMeas oscillates for two cycles
  if (Keep(1) > Keep(2)),
    if(Keep(2) < Keep(3)),
      if(Keep(3) > Keep(4)),
        if (Keep(4) < Keep(5)),
          kk = 0.9*kk;
        end;
      end;
    end;
  end;
  if (Keep(1) < Keep(2)),
    if(Keep(2) > Keep(3)),
      if(Keep(3) < Keep(4)),
        if (Keep(4) > Keep(5)),
          kk = 0.9*kk;
        end;
      end;
    end;
  end;

% Determine how well system is performing
  if (sign(max(OUT))==sign(min(OUT))),    %Range of output without zero
    APE(t) = sum(abs(OUT-O5)./abs(OUT))*100/P
  else                                    %Range of output with zero
    APE(t) = sum(abs(OUT-O5))/sum(abs(OUT))*100
  end;

end;
```

```
%
%  Functions Called by the Above Main Program
%

function y = deriva(PP,x)
%  Derivative of a first layer node wrt a.
%
a = PP(1);
b = PP(2);
c = PP(3);
y = 2*((1+(((x-c)/a)^2)^b)^(-2))*(((x-c)/a)^2)^(b-1)*((x-c)/a)^2/a*b;


function y = derivb(PP,x)
%  Derivative of a first layer node wrt b.
%
a = PP(1);
b = PP(2);
c = PP(3);
x = x + 1e-10;
y = real( -2*log((x-c)/a)*(((x-c)/a)^2)^b*(1+(((x-c)/a)^2)^b)^(-2));


function y = derivc(PP,x)
%  Derivative of a first layer node wrt c.
%
a = PP(1);
b = PP(2);
c = PP(3);
x = x + 1e-10;
y = 2*(1+(((x-c)/a)^2)^b)^(-2)*(((x-c)/a)^2)^(b-1)*((x-c)/a)*(b/a);


function y=member(A,b)
% Function used to determine if the element b is
%   in the vector A.
%
y=0;
for i=1:length(A),
  if (A(i) == b),
    y=1;
  end;
end;
```

APPENDIX C: GRAPHICAL RESULTS FROM THE SIMULATION

The following are the results for three examples used to illustrate how well the GNN program works. The first example uses as input data the raw data given in Table C.1. This data represents the output of an actual fuzzy control system that was designed by an expert [Ref. 1]. The shape of the target output is given in Figure C.1. The data was used for three different setups of the GNN, each one differing by the number of membership functions used for each input in the first layer. Runs using two, three, and four membership functions per input were verified and the final adjusted membership functions are shown. A measure of the error between the desired output and actual output in the form of an Average Percentage Error is also given. The shape of the initial membership functions are somewhat arbitrary, and were picked to give the most coverage to the input data.

The second example is an attempt to learn the input/output relationship given by Equation C.1. Similar to the first example, three different setups for the GNN were used. Results for these runs are given in Figures C.15 through C.22.

$$z = \left(3\exp\left(\frac{y}{10}\right) - 1\right) \cdot 15 \cdot \tanh\left(\frac{x}{2}\right) + \left(4 + \exp\left(\frac{y}{10}\right)\right) \cdot 8 \cdot \sin\frac{(x+4)\cdot\pi}{10} \qquad \text{(C.1)}$$

The final example involves three inputs and a single output with the relationship given by Equation C.2. Only two setups of the GNN were used. The results of the training of the membership functions are shown in Figures C.31 through C.36. Since this example involes four dimensions, the actual target surface cannot be shown.

$$output = \left(1 + x^{0.5} + y^{-1} + z^{-1.5}\right)^2 \qquad \text{(C.2)}$$

The error measure for all examples was an Average Percentage Error (APE) and is given by either Equation C.3, for the first two examples, or by Equation C.4 for the last example[1]. In these equations, P is the total number of training data samples.

$$APE_{1,2} = \frac{\sum_{i=1}^{P} |(OUTPUT_{desired} - OUTPUT_{actual}|}{\sum_{i=1}^{P} |OUTPUT_{desired}|} \cdot 100\%. \tag{C.3}$$

$$APE_3 = \frac{1}{P} \cdot \sum_{i=1}^{P} \frac{|OUTPUT_{desired} - OUTPUT_{actual}|}{|OUTPUT_{desired}|} \cdot 100\%. \tag{C.4}$$

[1]Which equation was used depended on the range of the output. If the output range included zero, as in the first two examples, the second equation was inappropriate, due to a possible singularity in the denominator. However, for other cases, the second equation would give a more accurate measure of actual error performance, and thus was used for the third example.

TABLE C.1: INPUT AND OUTPUT TRAINING DATA FOR EXAMPLE 1

X Input	Y Input	Output
-10.0000000000	-10.0000000000	-46.9562411241
-10.0000000000	-8.0000000000	-12.0169721617
-10.0000000000	-6.0000000000	-2.1875345706
-10.0000000000	-4.0000000000	-5.9453743885
-10.0000000000	-2.0000000000	-13.7846470479
-10.0000000000	0.0000000000	-22.5508919169
-10.0000000000	2.0000000000	-29.7566400578
-10.0000000000	4.0000000000	-16.6985904620
-10.0000000000	6.0000000000	-1.6064559431
-10.0000000000	8.0000000000	4.6515359660
-10.0000000000	10.0000000000	-4.8858288195
-8.0000000000	-10.0000000000	-41.4493837739
-8.0000000000	-8.0000000000	-9.1956603788
-8.0000000000	-6.0000000000	-0.6517689562
-8.0000000000	-4.0000000000	-4.7272998310
-8.0000000000	-2.0000000000	-12.5434162388
-8.0000000000	0.0000000000	-21.2002614155
-8.0000000000	2.0000000000	-28.4892287464
-8.0000000000	4.0000000000	-17.7389693395
-8.0000000000	6.0000000000	-4.6744935218
-8.0000000000	8.0000000000	0.6979161170
-8.0000000000	10.0000000000	-8.2415625506
-6.0000000000	-10.0000000000	-36.3992821301
-6.0000000000	-8.0000000000	-6.9164981277
-6.0000000000	-6.0000000000	0.3638556988
-6.0000000000	-4.0000000000	-3.9524544648
-6.0000000000	-2.0000000000	-11.6520872764
-6.0000000000	0.0000000000	-20.0993028579
-6.0000000000	2.0000000000	-27.3519750873
-6.0000000000	4.0000000000	-18.7093080582
-6.0000000000	6.0000000000	-7.6777917550
-6.0000000000	8.0000000000	-3.3034993487
-6.0000000000	10.0000000000	-11.7113843694
-4.0000000000	-10.0000000000	-32.2385453040
-4.0000000000	-8.0000000000	-5.6901860757
-4.0000000000	-6.0000000000	0.3703069449
-4.0000000000	-4.0000000000	-4.0373987614
-4.0000000000	-2.0000000000	-11.4395860949
-4.0000000000	0.0000000000	-19.4828305507
-4.0000000000	2.0000000000	-26.4668541073
-4.0000000000	4.0000000000	-19.5378103128
-4.0000000000	6.0000000000	-10.5466428752
-4.0000000000	8.0000000000	-7.3891898670
-4.0000000000	10.0000000000	-15.3968241994
-2.0000000000	-10.0000000000	-34.3541216014
-2.0000000000	-8.0000000000	-11.3438486194
-2.0000000000	-6.0000000000	-6.0586611219
-2.0000000000	-4.0000000000	-9.6148259808
-2.0000000000	-2.0000000000	-15.6372485606
-2.0000000000	0.0000000000	-22.1375784549
-2.0000000000	2.0000000000	-27.5568404601
-2.0000000000	4.0000000000	-20.6442158113
-2.0000000000	6.0000000000	-14.1912311753
-2.0000000000	8.0000000000	-13.6787441341
-2.0000000000	10.0000000000	-21.8877559979
0.0000000000	-10.0000000000	-37.4851910308
0.0000000000	-8.0000000000	-19.3801366601
0.0000000000	-6.0000000000	-15.1135884169
0.0000000000	-4.0000000000	-17.4100512058
0.0000000000	-2.0000000000	-21.4305474813
0.0000000000	0.0000000000	-25.6746105893
0.0000000000	2.0000000000	-28.5742541135
0.0000000000	4.0000000000	-19.2672215261
0.0000000000	6.0000000000	-14.5761498363
0.0000000000	8.0000000000	-17.1652439776
0.0000000000	10.0000000000	-26.2515682937
2.0000000000	-10.0000000000	-27.6106521916
2.0000000000	-8.0000000000	-14.1081409823
2.0000000000	-6.0000000000	-11.9567373160
2.0000000000	-4.0000000000	-14.6735316286
2.0000000000	-2.0000000000	-18.5134561779
2.0000000000	0.0000000000	-22.3774422146
2.0000000000	2.0000000000	-24.7828536400
2.0000000000	4.0000000000	-14.6612513476
2.0000000000	6.0000000000	-9.5725257096
2.0000000000	8.0000000000	-12.0462146156
2.0000000000	10.0000000000	-21.0165360981
4.0000000000	-10.0000000000	-16.4446809784
4.0000000000	-8.0000000000	-5.9262193228
4.0000000000	-6.0000000000	-5.3133096070
4.0000000000	-4.0000000000	-8.5339980452
4.0000000000	-2.0000000000	-12.5025954602
4.0000000000	0.0000000000	-16.3697706307
4.0000000000	2.0000000000	-18.6588715519
4.0000000000	4.0000000000	-6.9513989790
4.0000000000	6.0000000000	0.9403010692
4.0000000000	8.0000000000	0.9988191873
4.0000000000	10.0000000000	-7.1112553691
6.0000000000	-10.0000000000	-11.8954608721
6.0000000000	-8.0000000000	-0.1571426971
6.0000000000	-6.0000000000	1.1110315051
6.0000000000	-4.0000000000	-1.8670036149
6.0000000000	-2.0000000000	-5.7410877324
6.0000000000	0.0000000000	-9.5363921647
6.0000000000	2.0000000000	-11.4318942123
6.0000000000	4.0000000000	5.0091276040
6.0000000000	6.0000000000	19.1795514756
6.0000000000	8.0000000000	24.0038866997
6.0000000000	10.0000000000	16.7180909374
8.0000000000	-10.0000000000	-14.4788756797
8.0000000000	-8.0000000000	1.4387587193
8.0000000000	-6.0000000000	4.9242408368
8.0000000000	-4.0000000000	2.7399064339
8.0000000000	-2.0000000000	-0.8525463618
8.0000000000	0.0000000000	-4.4829829884
8.0000000000	2.0000000000	-5.7233342568
8.0000000000	4.0000000000	17.2356958553
8.0000000000	6.0000000000	38.2625805935
8.0000000000	8.0000000000	47.3029247940
8.0000000000	10.0000000000	39.7872388333
10.0000000000	-10.0000000000	-19.1246911365
10.0000000000	-8.0000000000	1.1977401978
10.0000000000	-6.0000000000	7.0189003305
10.0000000000	-4.0000000000	5.6700498239
10.0000000000	-2.0000000000	2.3752044148
10.0000000000	0.0000000000	-1.0754895047
10.0000000000	2.0000000000	-1.6192367099
10.0000000000	4.0000000000	27.6685627469
10.0000000000	6.0000000000	54.3576306394
10.0000000000	8.0000000000	66.1873838293
10.0000000000	10.0000000000	57.6934361793

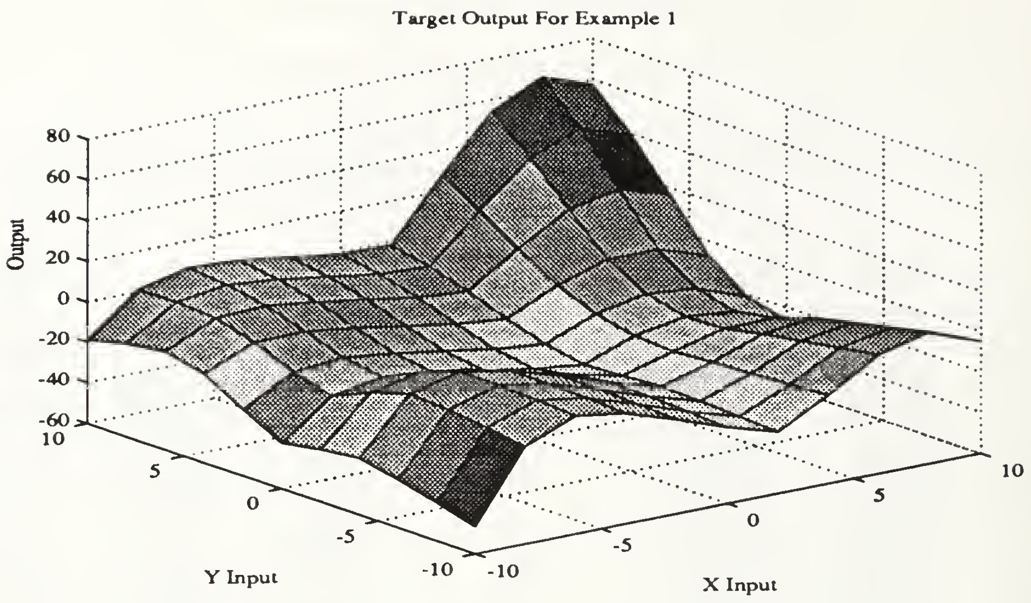

Figure C.1: Target Output For Example 1

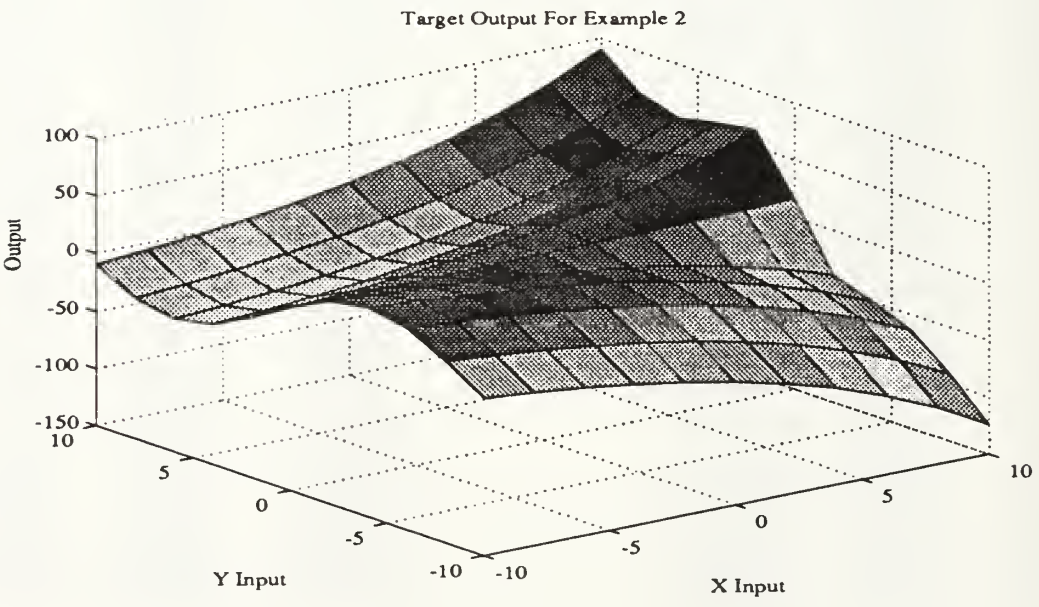

Figure C.2: Target Output For Example 2

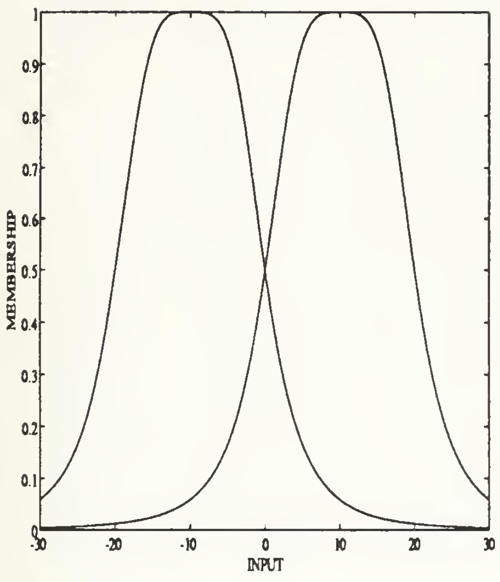

Figure C.3: Initial Membership Functions (Ex. 1, Two MF's)

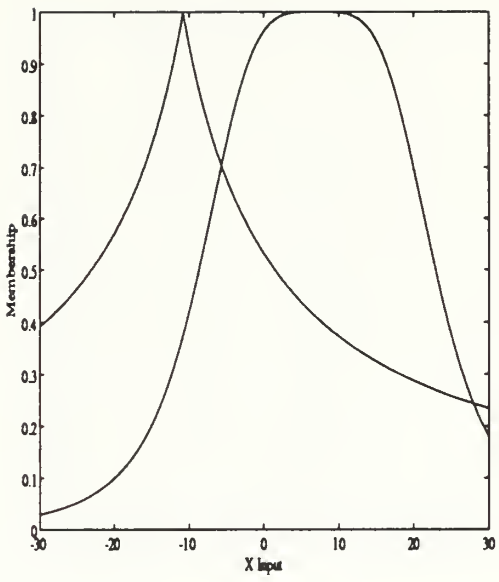

Figure C.5: Final Y Membership Functions (Ex. 1, Two MF's)

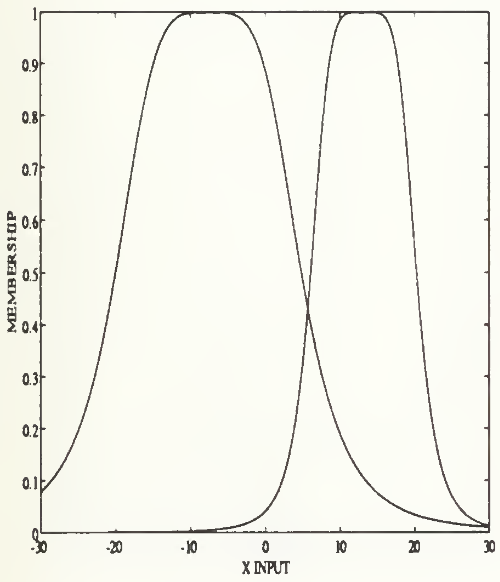

Figure C.4: Final X Membership Functions (Ex. 1, Two MF's)

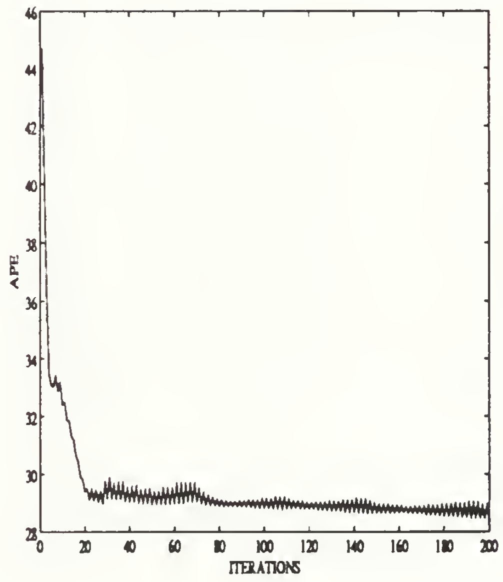

Figure C.6: Average Percentage Error (Ex. 1, Two MF's)

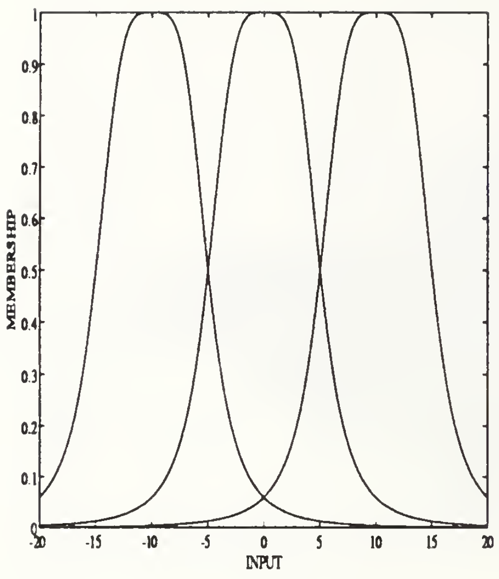

Figure C.7: Initial Membership Functions (Ex. 1, Three MF's)

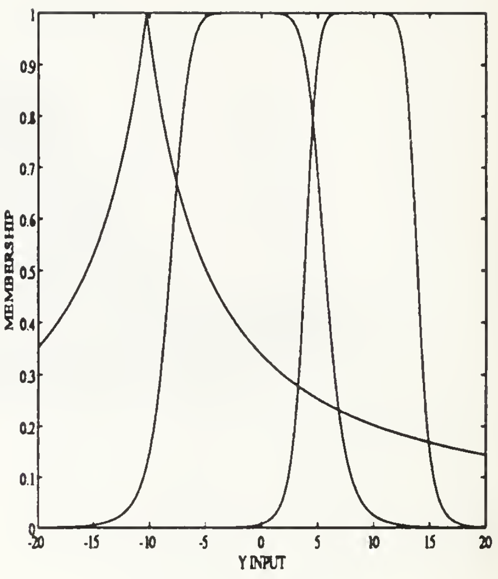

Figure C.9: Final Y Membership Functions (Ex. 1, Three MF's)

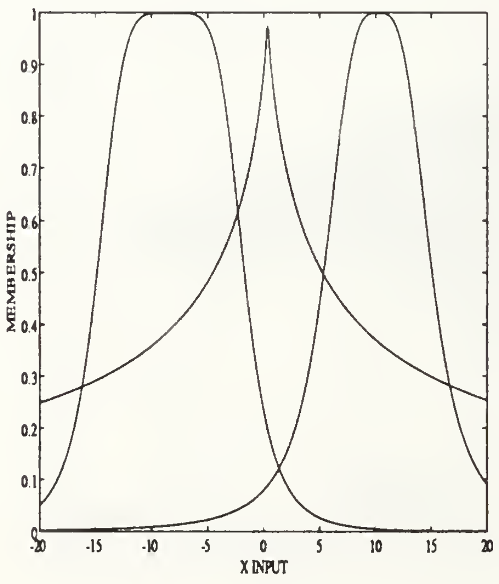

Figure C.8: Final X Membership Functions (Ex. 1, Three MF's)

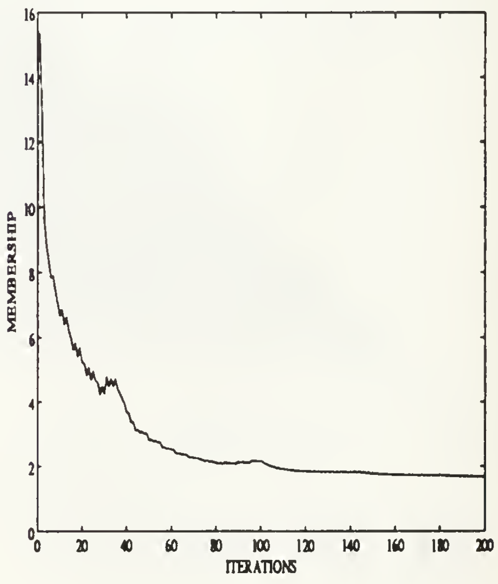

Figure C.10: Average Percentage Error (Ex. 1, Three MF's)

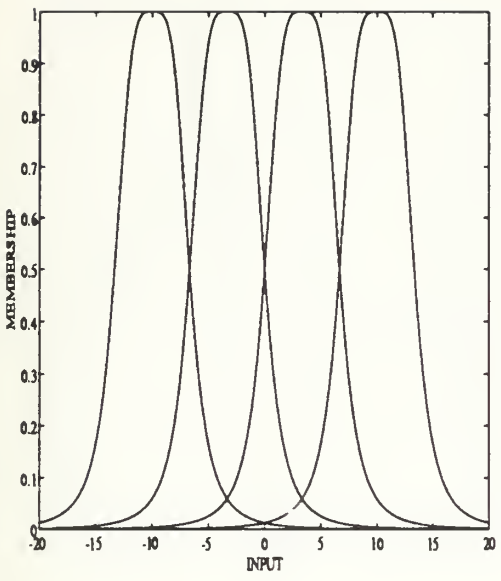

Figure C.11: Initial Membership Functions (Ex. 1, Four MF's)

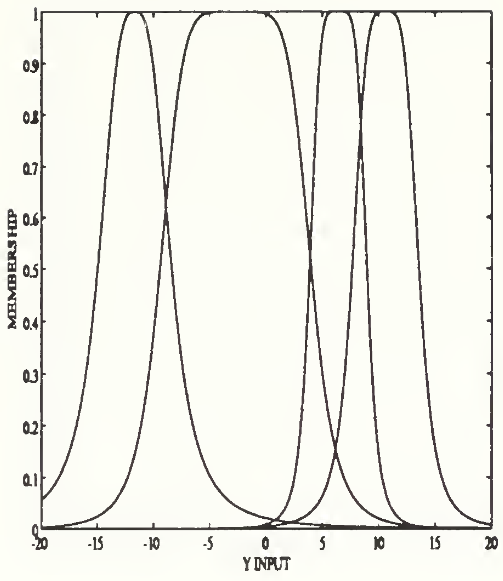

Figure C.13: Final Y Membership Functions (Ex. 1, Four MF's)

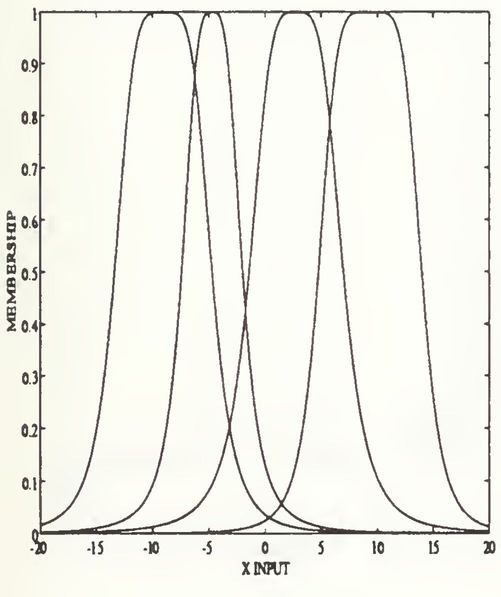

Figure C.12: Final X Membership Functions (Ex. 1, Four MF's)

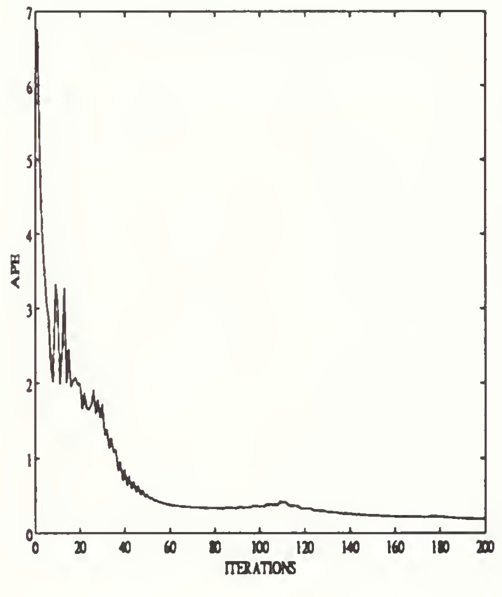

Figure C.14: Average Percentage Error (Ex. 1, Four MF's)

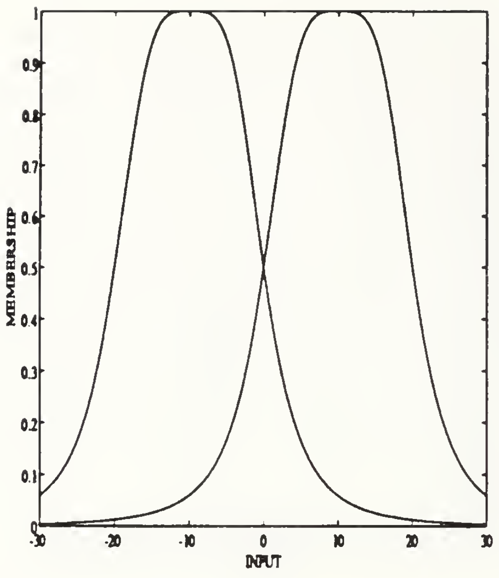

Figure C.15: Initial Membership Functions (Ex. 2, Two MF's)

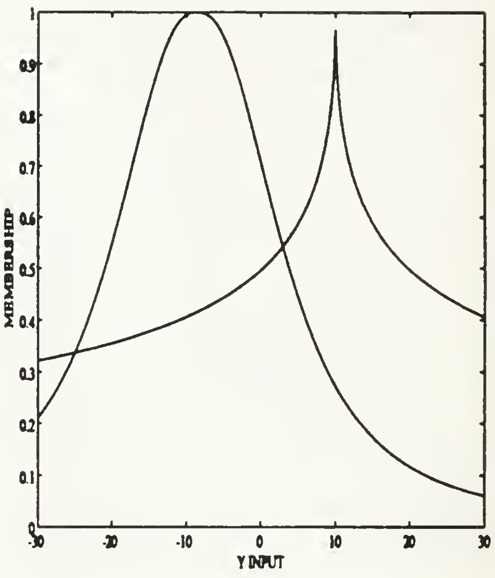

Figure C.17: Final Y Membership Functions (Ex. 2, Two MF's)

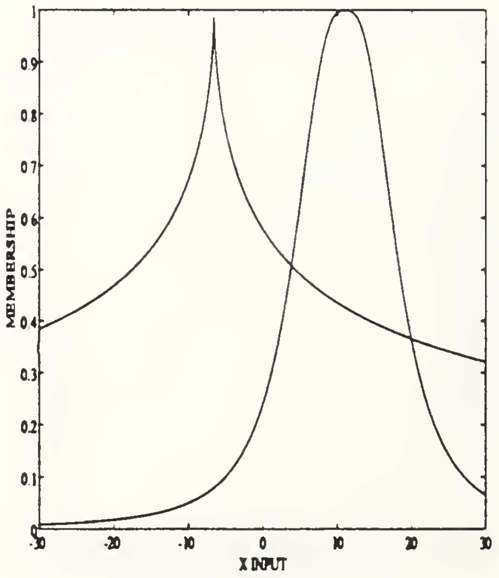

Figure C.16: Final X Membership Functions (Ex. 2, Two MF's)

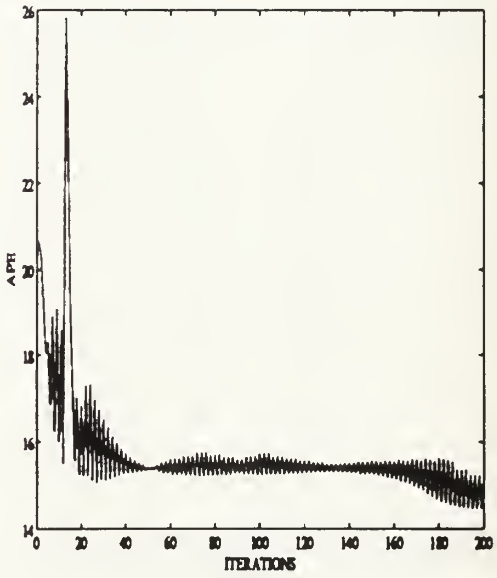

Figure C.18: Average Percentage Error (Ex. 2, Two MF's)

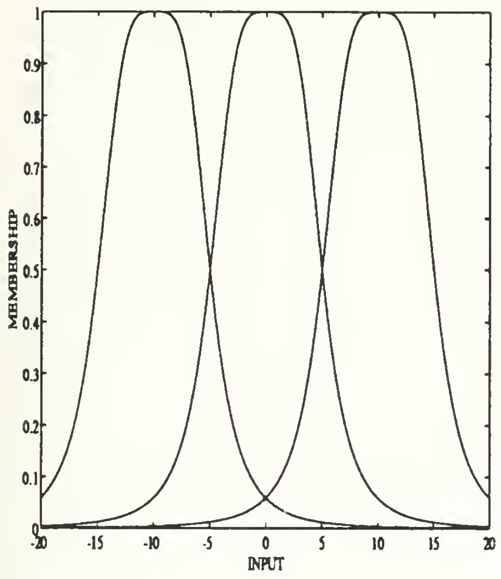

Figure C.19: Initial Membership Functions (Ex. 2, Three MF's)

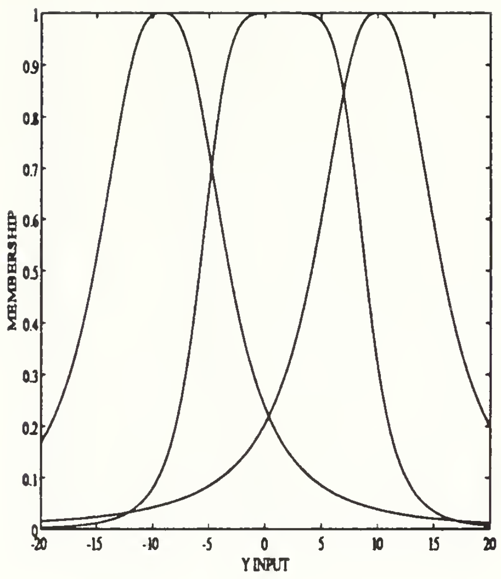

Figure C.21: Final Y Membership Functions (Ex. 2, Three MF's)

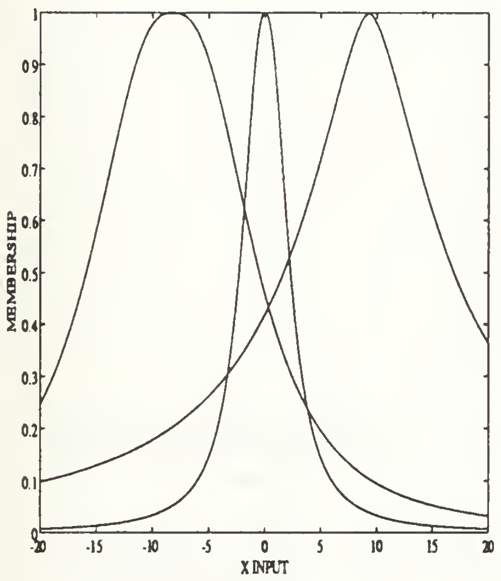

Figure C.20: Final X Membership Functions (Ex. 2, Three MF's)

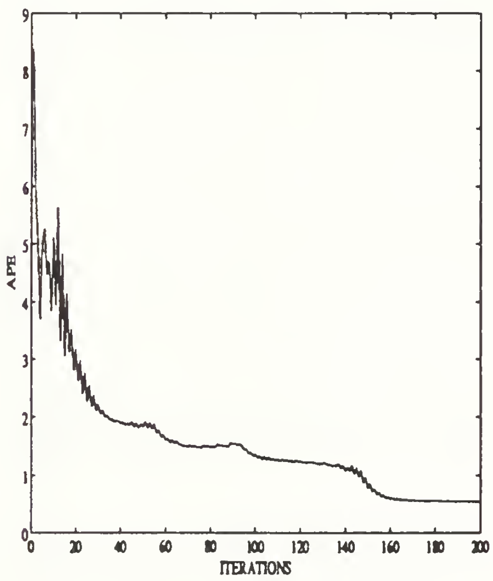

Figure C.22: Average Percentage Error (Ex. 2, Three MF's)

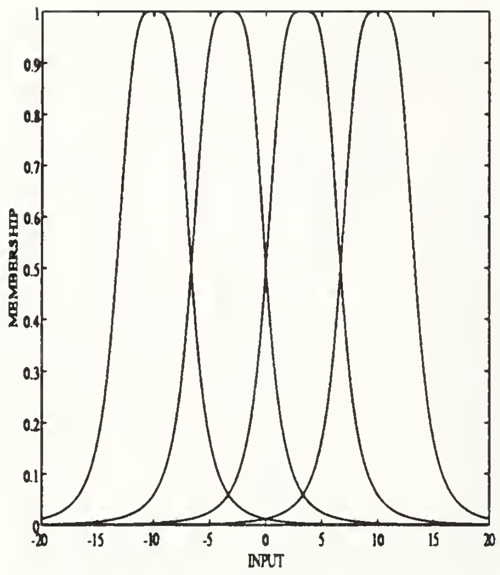

Figure C.23: Initial Membership Functions (Ex. 2, Four MF's)

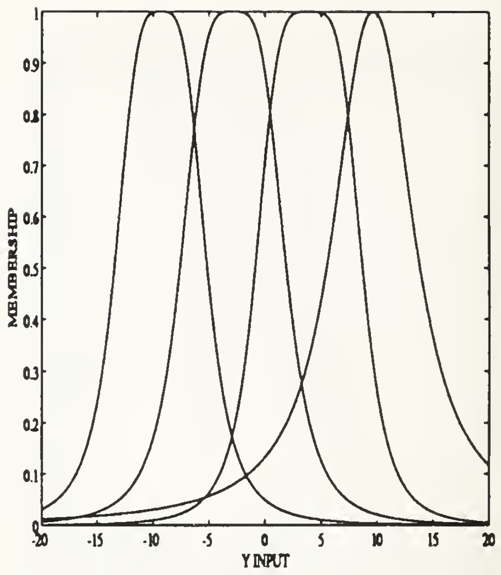

Figure C.25: Final Y Membership Functions (Ex. 2, Four MF's)

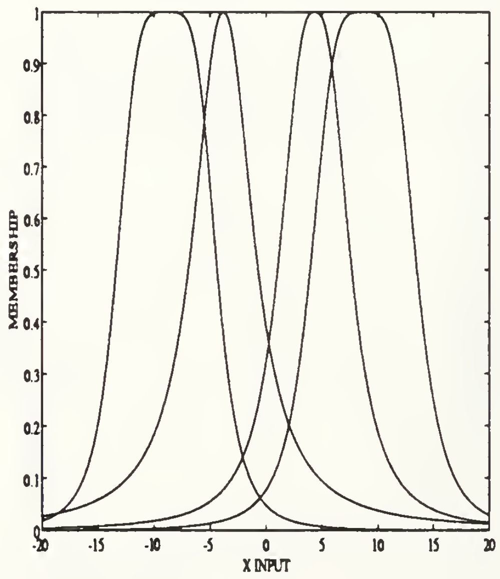

Figure C.24: Final X Membership Functions (Ex. 2, Four MF's)

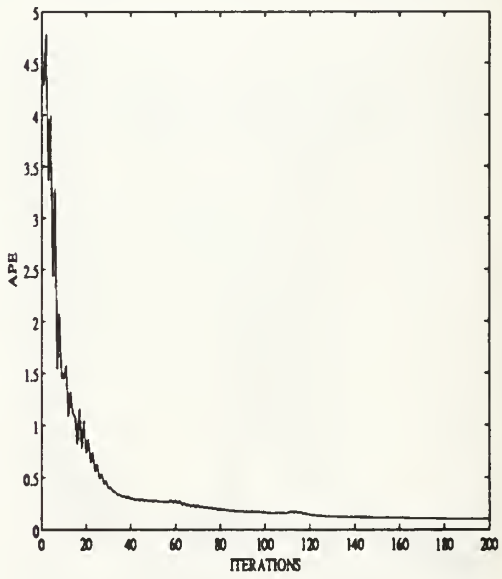

Figure C.26: Average Percentage Error (Ex. 2, Four MF's)

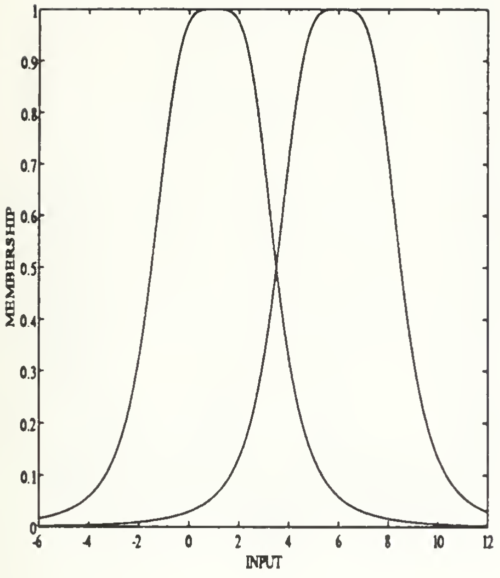

Figure C.27: Initial Membership Functions (Ex. 3, Two MF's)

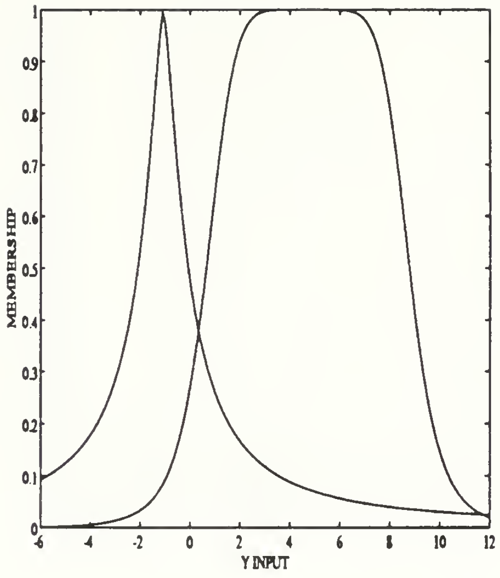

Figure C.29: Final Y Membership Functions (Ex. 3, Two MF's)

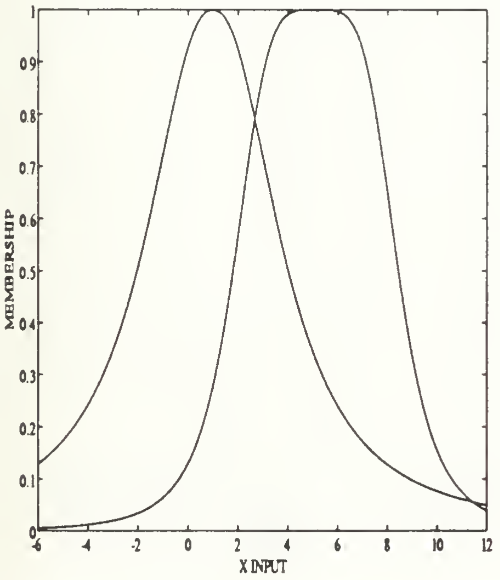

Figure C.28: Final X Membership Functions (Ex. 3, Two MF's)

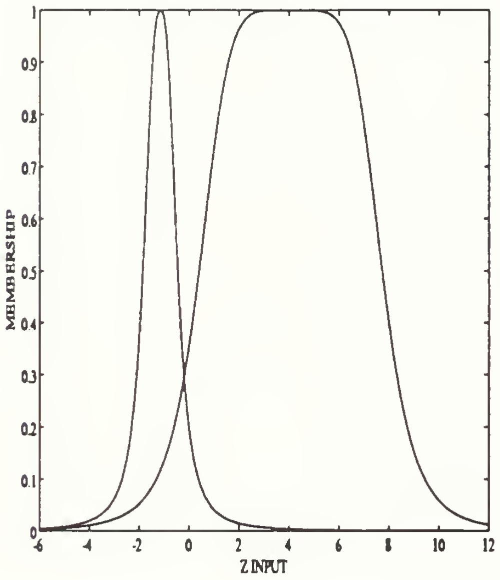

Figure C.30: Final Z Membership Functions (Ex. 3, Two MF's)

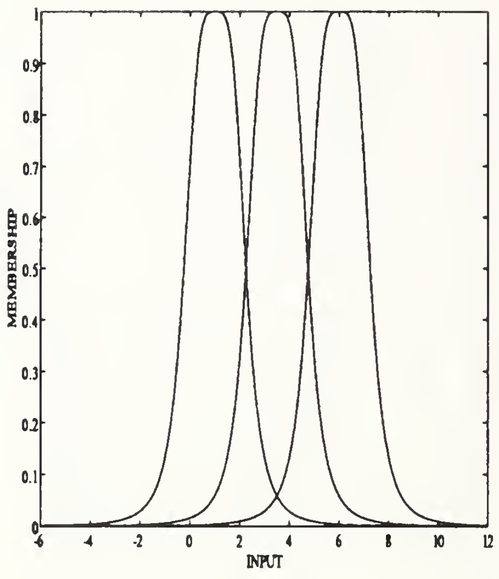

Figure C.31: Initial Membership Functions (Ex. 3, Three MF's)

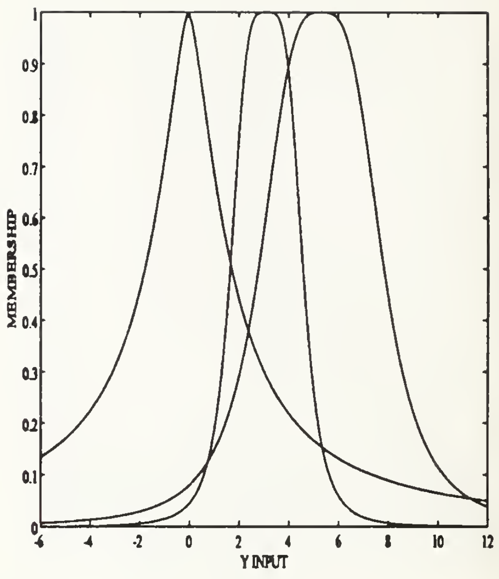

Figure C.33: Final Y Membership Functions (Ex. 3, Three MF's)

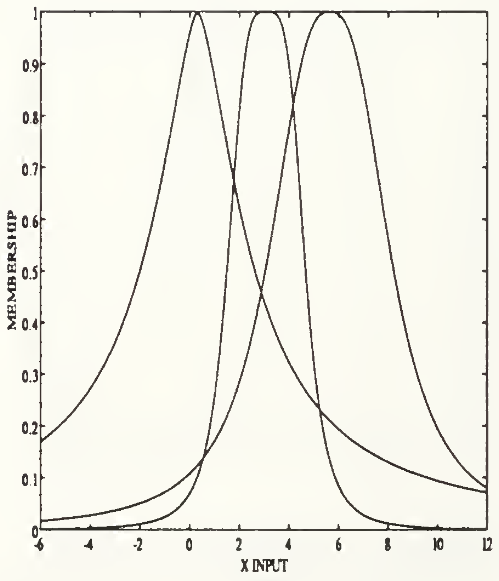

Figure C.32: Final X Membership Functions (Ex. 3, Three MF's)

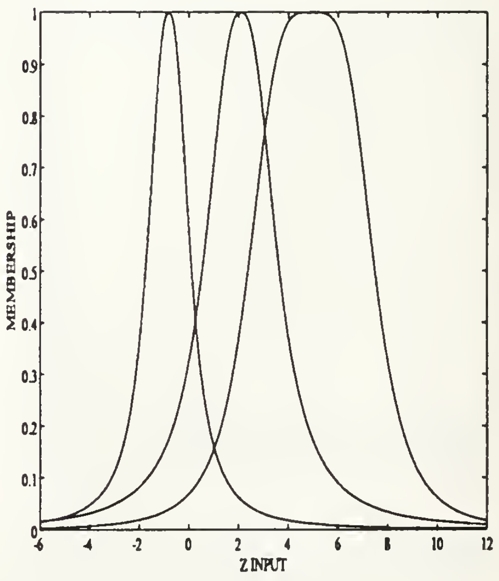

Figure C.34: Final Z Membership Functions (Ex. 3, Three MF's)

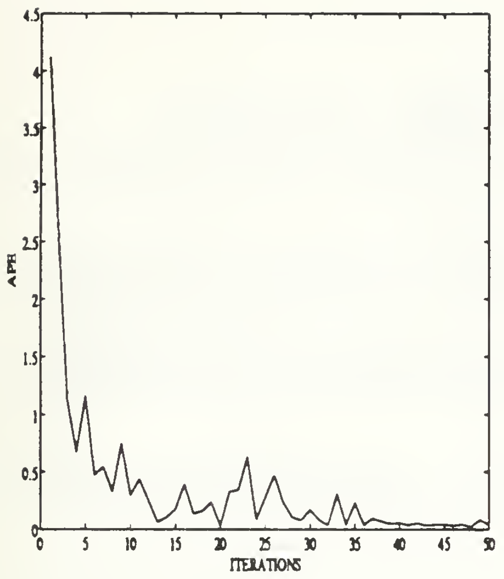

Figure C.35: Average Percentage Error (Ex. 3, Two MF's)

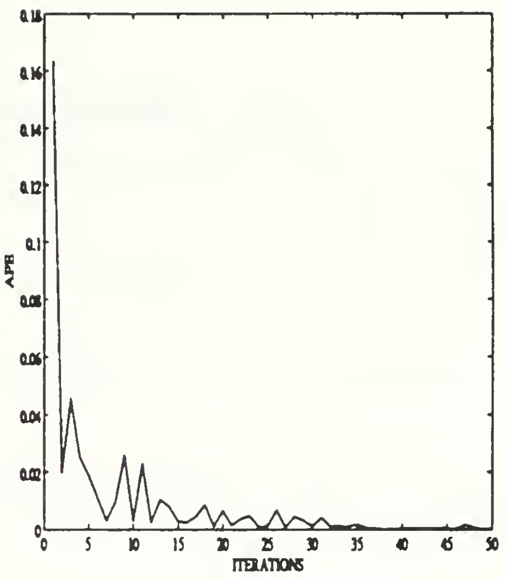

Figure C.36: Average Percentage Error (Ex. 3, Three MF's)

REFERENCES

1. Jyh-Shing Jang, "Fuzzy Modeling Using Generalized Neural Networks and Kalman Filter Algorithm," AAAI 1991, pp. 762-767.

2. Hideyuki Takagi and Isao Hayashi, "NN-Driven Fuzzy Reasoning," *International Journal of Approximate Reasoning*, Vol. 5, 1991, pp. 191-212.

3. Ching-Teng Lin and C.S. George Lee, "Neural-Network-Based Fuzzy Logic Control and Decision System," *IEEE Trans. Comp.*, Vol. 40, No. 12, December 1991, pp. 1320-1336.

4. Ronald R. Yager, "Modeling and Formulating Fuzzy Knowledge Bases Using Neural Networks," Technical Report #MII-1111, Iona College, New Rochelle, NY, 1991.

5. Chuen-Chien Lee, "A Self-Learning Rule-Based Controller Employing Approximate Reasoning and Neural Net Concepts," *International Journal of Intelligent Systems*, Vol. 6, 1991, pp. 71-93.

6. Ichiro Enbutsu, Kenii Baba, and Naoki Hara, "Fuzzy Rule Extraction From a Multilayered Neural Network" In *IEEE INNS International Joint Conference on Neural Networks* edited by I.R. Goodman *et al.*, Vol II, pp. 461-465.

7. Michio Sugeno, "An Introductory Survey of Fuzzy Control," *Information Sciences*, Vol. 36, 1985, pp. 59-83.

8. Chuen-Chien Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller—Part I & II," *IEEE Trans. Syst., Man, and Cybern.*, Vol. 20, No. 2, 1990, pp. 404-435.

9. Richard P. Lippman, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, April 1987, pp. 4-22.

10. NeuralWare, Inc., *Neural Computing*, 1991.

11. G. Cybenko, "Continuous Value Neural Networks With Two Hidden Layers Are Sufficient," *Math Contr. Signal and Sys.*, Vol. 2, 1989, pp. 303-314.

12. Steven F. Zornetzer, Joel L. Davis, and Clifford Lau (eds.), *An Introduction to Neural and Electronic Networks,*, Academic Press:San Diego, 1990.

13. Dong-Hui Li and François E. Cellier, "Fuzzy Measures in Inductive Reasoning," In *Proceedings of the 1990 Winter Simulation Conference*, 1990, pp. 527-538.

14. M.M. Gupta and J. Qi, "Connectives (And,Or,Not) and T-Operators in Fuzzy Reasoning," In *Conditional Logic in Expert Systems*, 1991, pp. 211-227.

INITIAL DISTRIBUTION LIST

		No. of Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5002	1
4.	Professor Chyan Yang, Code EC/Ya Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	2
5.	Professor Jon T. Butler, EC/Bu Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	2
6.	Lt. Billy E. Hudgins 914 Ninth Avenue Albany, GA 31701	2